Structures and Derived Types

The *components* of a *structure* do not have to have the same type, whereas the *elements* of an *array* must all have the same type.

The components of a structure are referenced by a *name*, whereas the elements of an array are referenced by an integer value.

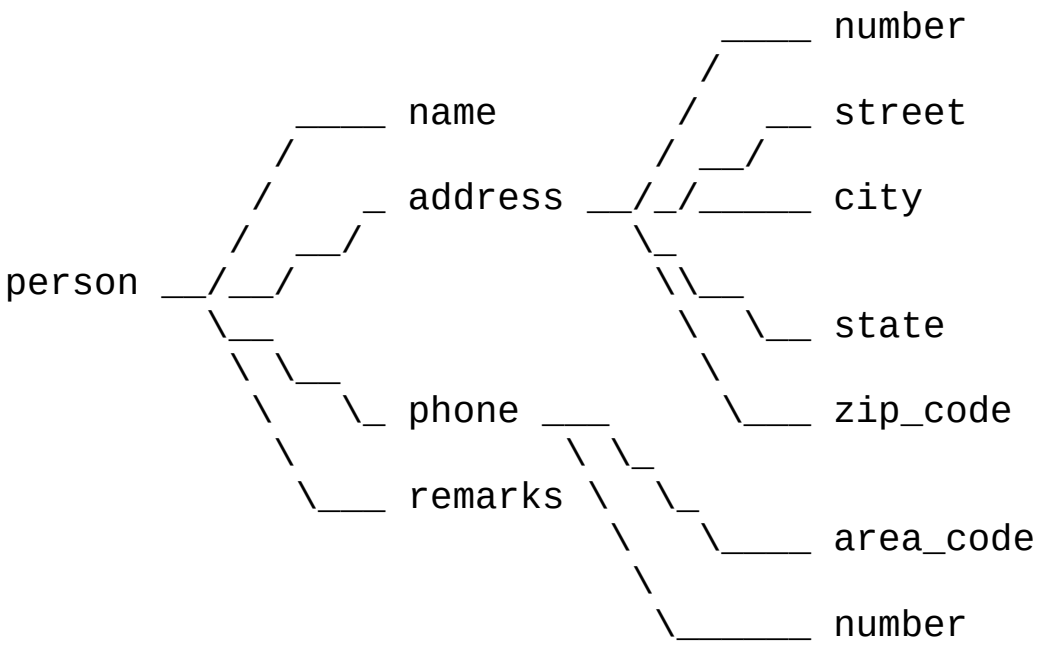
Structures may be nested (i.e., a component of a structure may be a structure or an array). The elements of an array may be structures (all of the same type).

Structures can be used to build recursive data types, such as linked lists and trees.

Next slide

person

Consider the following structure:



Previous slide Next slide

Defining Derived Types

To use structures, a new type, *a derived type*, must be defined. Then variables that are structures are declared to be that type.

```
type phone_type
   integer :: area_code, number
end type phone_type

type address_type
   integer :: number
   character (len = 30) :: street, city
   character (len = 2) :: state
   integer :: zip_code
end type address_type
```

Previous slide Next slide

```
type person_type
   character (len = 40) :: name
   type (address_type) :: address
   type (phone_type) :: phone
   character (len = 100) :: remarks
end type person_type
```

Learn more about derived type definitions.

Previous slide Next slide

Declaring Structures

```
type (person_type) :: joan

type (person_type), dimension (1000) ::  &
      black_book
```

Previous slide Next slide

Referencing Structure Components

Joan's address is

```
joan % address
```

The area code of the third person in the little black book is:

```
black_book (3) % phone % area_code
```

Learn more about structure components.

Previous slide Next slide

Structure Constructor

Each derived type has a structure constructor that builds values of that type.

```
joan % phone = phone_type (505, 2750800)

joan = person_type ("Joan Doe",  &
       john % address,  &
       phone_type (505, fax_number - 1),  &
       "Same address as husband John")
```

Learn more about structure constructors.

Previous slide Next slide

Operations on Structures

The only defined operations on structures are assignment and input/output.

```
black_book (7) = joan
```

For formatted I/O, there must be an edit descriptor for each ultimate component.

```
print "(i3, i7)", joan % phone
```

Previous slide Next slide

Default Initialization (F95)

A default value may be given to a component of a derived type. When a structure of this type is declared or allocated, the component is initialized.

```
program test_def_init

   implicit none

   type string
      integer :: length = 0
      character (len=99) :: s = ""
   end type string

   type (string) :: s1, sa(99)

   print *, s1%length
   sa(93) = string(sa(93)%length + 3, "xxx")
   print *, sa(93)

end program test_def_init
```

Learn more about default initialization.

Previous slide Next slide

Type Equivalence

In order for actual and dummy arguments of derived type to match correctly, they must be the same type.

Unless they are `sequence` types (see below), this means they must be the same type. The only practical way to achieve this is to put the type definition in a module and use the module in both the calling and called procedures.

Previous slide Next slide

The sequence Statement

A derived type definition may contain the `sequence` statement, consisting of just the keyword. This indicates that the storage for the components of the structure being defined are laid out in a specific way, allowing structures of that type to work with other features depending on storage layout, such as `equivalence` and `common`.

```
type polar
   sequence
   real rho, theta
end type polar
```

Previous slide